Moj računalnik je lahko tudi kalkulator

Če poženemo Python dobimo ... kalkulator.

```
>>> 1 + 1
2
>>> 2 * 3
6
>>> 1+2 * 3+1
8
```

Nič posebnega. Temu, kar smo vpisovali, pravimo *izraz*. Pri tem predmetu se sicer ne bomo ukvarjali s pravorečjem; teoretiki nas učijo o razliki med *izrazom*, *stavkom* in bogvečem še, sam pa se tule enkrat za vselej opravičujem, ker bom dal prednost praksi, ne terminologiji. Še definicijo pomena besede *izraz* bom prepustil kolegu Slivniku, ki bo nekatere od vas v tretjem letniku učil zanimivo snov o jezikih, prevajalnikih in sploh vsem).

Izraz bo za nas pač *nekaj, kar se da izračunati* (kar je ravnokar zaropotalo, je neki Alan Turing, ki se je z jabolkom v ustih obrnil v grobu). In gornje stvari se očitno dajo izračunati. In niti niso preveč zanimive.

Razen, morda, zadnjega izraza. V izrazih lahko uporabljamo presledke in to moramo - kot tudi sicer - početi po občutku. V zadnjem izrazu tega nismo počeli, zato je zapis zavajajoč. Python je dovolj pameten, da ve, da ima *operator* (še ena beseda, ki si jo zapomnimo!) množenja prednost pred operatorjem seštevanja. Kar smo napisali, se računa enako, kot če bi rekli

```
1+2*3+1
```

ali

```
>>> 1 + 2*3 + 1
```

ali

```
>>> 1 + 2 * 3 + 1
```

Nekoč se bomo dogovorili za zadnje, dotlej pa jih pišite, kakor hočete, le nekaj se zmenimo: nikoli jih ne dajajte na začetek. Dokler vam naslednji teden ne bom rekel, da jih morate. Boste videli, zakaj.

Kakšni operatorji so nam še na voljo? Očitna sta še `/` za deljenje in `-` za odštevanje. Dvojna zvezdica, `**`, pomeni potenciranje (mednju ne smemo napisati presledka, `**` je kot, recimo, ena *beseda*). Operator `%` izračuna ostanek po deljenju.

```
>>> 5 ** 2
25
>>> 3 ** 4
81
>>> 13 % 5
```

```
3
```

Potenciranje ima prednost pred vsem, kar poznamo doslej. Kadar je treba, pa lahko uporabimo oklepaje.

```
>>> (4 + 5) * 4
36
```

Če ostane kak oklepaj odprt, Python ne izračuna izraza, saj ve, da ga še nismo dokončali. Namesto `>>>` pokaže tri pike in dovoli, da nadaljujemo izraz v naslednji vrsti.

```
>>> (4 + 2 * (3
... + 8)-
... 2)
24
```

Tule ni lepo, a kdaj drugič nam bo prišlo še zelo prav, boste videli.

Pri množenju je nujno uporabiti zvezdico. Se pravi, pisati moramo `7 * (2 + 3)` in ne `7(2 + 3)`.

Za deljenje imamo poleg operatorja `/` tudi `//`, ki deli celoštevilsko.

```
>>> 4.5 // 1.2
3.0
>>> 7 // 2
3
```

1.2 gre v 4.5 trikrat ... in še malo ostane. A celoštevilskega deljenja ostanek ne zanima.

Pozornejšemu je padlo v oči še nekaj zanimivega: če delimo 4.5 z 1.2, dobimo 3.0, če delimo 7 z 2, pa 3. Za prvo predavanje bi lahko to tudi preskočili, a se vseeno pomudimo: Python loči med celimi in necelimi števili. Rezultat deljenja necelih števil je necelo število; slučajno je ravno okroglo (3), vendar Python še vedno ve, da gre za število, ki bi lahko bilo necelo (čeprav slučajno ni). Rezultat celoštevilskega deljenja celih števil pa je celo število, zato ga Python tudi izpiše brez decimalk. Podobno je s seštevanjem.

```
>>> 2.3 + 1.7
4.0
>>> 2 + 2
4
```

Če seštejemo realni števili 2.3 in 1.7 dobimo realno število 4.0. Če seštejemo celi števili 2 in 2, dobimo celo število 4.

Kako hitro napredujemo! To, kar smo pravkar spoznali, so "podatkovni tipi". Točneje, spoznali smo dva podatkovna tipa *cela števila* in *števila s plavajočo vejico*. V angleščini se jima reče *integer* in *floating point number* v Pythonu pa `int` in `float`. Odkod ta čudna imena boste izvedeli pri kakem drugem predmetu.

Vsaka reč v Pythonu je reč nekega tipa, in če je ta reč število, je bodisi tipa `int` bodisi `float`. (So števila lahko še kakega drugega tipa? Lahko, nekateri jeziki imajo celo kupe številskih tipov. Vendar nas za zdaj ne brigajo.)

Neučakanega študenta morda pograbila radovednost. Kateri podatkovni tipi pa še obstajajo - razen številskih? Le malo naj počaka, kmalu bodo na vrsti.

Preden gremo naprej, samo opozorimo, kaj vas čaka v večini drugih jezikov: v skoraj vseh drugih jezikih deljenje celih števil vrača celo število; v Javi, ki se jo boste učili v drugem semestru, bo 7 / 2 enako 3. Obratno opozorilo, seveda, velja za tiste, ki že znate programirati in ste morda vajeni drugače: v Pythonu izraz 7 / 2 vrne 3.5. Je to pametno? Najbrž je: ko delimo, večinoma hočemo "pravo" deljenje in le redko celoštevilskega. Še več, pri programiranju velikokrat naredimo napako, ko brezskrbno delimo, ne da bi pomislili na to, s kakšnimi števili - celimi ali ne - delamo. Avtorji Pythona so se zato odločili, naj bo deljenje vedno "pravo", kadar hočemo celoštevilsko, pa moramo to posebej povedati tako, da namesto `/` uporabimo `//`.

O obliki teh zapiskov

Gornja komunikacija s Pythonom je oblikovana, kot da poženemo Python iz ukazne vrstice. Python sprašuje z `>>>` in potem odgovarja.

Ti zapiski so pripravljeni z okoljem Jupyter Notebook. Tam vrstice, v katerih vnašamo ukaze Pythonu niso označene z `>>>`, temveč z `In`, njegovi odgovori pa z `Out`. Odslej bodo izpisi v takšni obliki. Če si namestite Jupypter Notebook, lahko prenesete te zapiske na svoj računalnik in izvajate dele kot, ki so v njem. Predlagam, da to poskusite in tudi počnete.

Moj računalnik je lahko tudi kalkulator - s spominom

Izračunali bomo, koliko je 2 + 3 in rekli računalniku, naj si to zapomni.

```
x = 2 + 3
```

Temu, kar smo napisali tu, pravimo *prireditveni stavek*, saj smo z njim x-u priredili vrednost izraza 2+3.

Tole zdaj je najbolj pomembna stvar v današnjem predavanju. Ne spreglejte je, čeprav je slišati trivialna, da ne bo kdaj kasneje postala kamen spotike (predvsem tistim, ki že znate programirati in si pod prirejanjem predstavljate nekaj drugega, kot v resnici je).

Da hočemo nekaj prirejati, povemo z enačajem (očitno). Na njegovi desni je nek izraz - torej nekaj, kar se da izračunati. Včasih bo to `2 + 3`, včasih bo kaj veliko bolj zapletenega, velikokrat pa bo na desni strani samo številka, kot, recimo, v prirejanju `x = 42`. Python bo, tako kot v našem dosedanjem igranju z njim, izračunal tisto na desni in dobil `5` ali `42` ali karkoli že.

Na levi strani enačaja je neko ime. Python bo temu imenu (recimo `x`) priredil tisto, kar je izračunal.

OK? Prireditveni stavek priredi imenu na levi strani enačaja rezultat izraza na desni.

Levo in desno od enačaja praviloma pišemo presledke, zaradi preglednosti.

Python tokrat ni izpisal ničesar v odgovor. Rezultat si je le zapomnil, shranil ga je pod imenom `x`. Kaj lahko počnemo s tem `x`? Lahko ga uporabljamo v drugih izrazih.

```
x + 7
```

```
12
```

```
x ** 2
```

```
25
```

```
13 % x
```

```
3
```

```
x
```

```
5
```

Kadar rečem `x`, Python poišče, tisto, kar je priredil `x`-u. Če smo `x`-u priredili 5 in rečemo `x + 7`, je to isto, kot če bi rekli `5 + 7`.

Temu `x` Slovenci ponosno rečemo *spremenljivka*. Angleži temu namreč pravijo *variable*, Madžari pa változó. (Predvsem slednje ni posebej pomembno in ne pride v poštev kot izpitno vprašanje, čeprav ni nič narobe, če veste. Koristi pa po drugi strani tudi nobene. ;))

Spremenljivko lahko seveda uporabljamo tudi za računanje novih spremenljivk.

```
y = x + 2
y
```

```
7
```

Spremenljivka pri programiranju (v večini jezikov) ne pomeni istega kot v matematiki. Spremenljivke v matematiki se, roko na srce, pravzaprav ne spreminjajo. V matematiki x ne more biti v eni vrstici 5, v naslednji pa 8. Pri programiranju pa lahko.

```
x = 5
x
```

```
5
```

```
x = 8
x
```

```
8
```

Še huje. Če matematiki ne bi znali programirati (pa navadno znajo in to dobro), bi jih utegnilo pretresti tole:

```
x = 5
x = x + 2
x
```

```
7
```

Kako je x lahko enak $x + 2$? Saj ni. Ta enačaj ne predstavlja enakosti, kot v matematiki, temveč prirejanje. Ja? V drugi vrstici Python izračuna vrednost izraza x + 2, to je, 7, in to priredi imenu, spremenljivki `x`. Izraz `x = x + 2` torej pomeni, preprosto, povečaj x za 2. (Napišite kaj takega pred prof. Fijavžem, če si upate!)

Tiste, ki že znajo vsaj malo programirati v kakem drugem jeziku, je morda zmotilo, da spremenljivk nismo nikjer deklarirali. (Tisti, ki jim beseda *deklarirati* ne pomeni ničesar, naj ostanejo v umestni nevednosti.) V Pythonu tega (skoraj) ne moremo narediti. Spremenljivka (točneje, ime) se pojavi, ko jo uporabimo, in izgine, ko je ne potrebujemo več. Ali je to dobro ali slabo, si niti teoretiki niso edini. Niti, ali je to dobro ali slabo za začetnika, ne.

Tem, ki že znajo programirati - posebej, če so uporabljali kak C# ali Javo - povejmo še, da se Pythonove spremenljivke tudi sicer bistevno razlikujejo od spremenljivk, ki so jih vajeni od ondod. Predvsem ni dobro, da so vas najbrž učili, da so "*spremenljivke kot nekakšne škatlice*". Za začetek bi bilo veliko lepše, če bi jim rekli *ime*, ne *spremenljivka*. Tudi Python jim reče *name*. A navada je železna srajca in vsi govorimo o *spremenljivkah*. Pomembno pa je tole: če imamo

```
x = 42
y = x
```

imamo v resnici eno samo škatlico (njena vsebina je število 42) z dvema imenoma, `x` in `y`. Več o tem bomo izvedeli kasneje, ko bomo to zmožnejši prebavljati in bomo videli tudi konkretno zmotnost ideje, da je "*spremenljivka kot nekakšna škatlica*".

Ostali, tisti, ki niso še nikoli programirali, pa se sprašujejo nekaj drugega. Kakšna so lahko imena spremenljivk? Vedno le ena črka? Angleške abecede?

Takole: imena (angleško govoreči jim pravijo *identifier*) so lahko poljubno dolga. Vsebujejo lahko črke angleške abecede, številke in podčrtaj, `_`, vendar se morajo začeti s črko ali podčrtajem, s številko pa ne. Python, kot skoraj vsi drugi jeziki, razlikuje med malimi in velikimi črkami: `x` in isto kot `X`. (V resnici smemo uporabljati tudi šumnike in kitajske pismenke, vendar se tega ne navadite, ker je nezaželeno in ker tega ne boste mogli početi v skoraj nobenem drugem jeziku kot v Pythonu, zato naj vam ne pride v kri.)

Poleg tega obstaja še par dogovorov. Vsak jezik ima namreč poleg obveznih

pravil tudi želeni slog. (Razen nekaterih, ki jih imajo več; najbolj notorično brezvladje je najbrž v C in C++.) Predvsem pa - in to vas bo kot računalnikarje prej ko slej zadelo - vsako resno podjetje določi pravila oblikovanja kode. Google, recimo, jih je predpisal za vse jezike, ki jih pogosteje uporablja (Google Style Guides). Vsi, ki programirajo v tem podjetju, se morajo teh pravil držati.

1. **Imena naj se vedno začnejo z malo črko.** Nekateri ste se učili drugih jezikov, kjer so navade drugačne. Ali pa so vas učili Python učitelji, ki nikoli niso brali pravil za Python. Kasneje bomo spoznali določene stvari, ki jih poimenujemo z veliko začetnico; dotlej pa - mala.

2. **Če je ime sestavljeno iz več besed, jih ločimo s podčrtajem,** , na primer `velikost_cevlja`. Obliko `velikostCevlja` boste videli le v starejših programih v Pythonu.

3. **Ime naj pove vsebino, ne vrste spremenljivke,** npr. `pospesek = 9.8`. Spremenljivke ne poimenujte `stevilka`, ceprav morda res vsebuje številko. Ko bomo začeli uporabljati sezname, vas bo veliko uporabljalo ime `seznam`, in če bodo v programu trije seznami, jih obste poimenovali `seznam1`, `seznam2`, `seznam3`. Da, to vam bo zelo pomagalo razumeti, kaj ste shranili kam. To je približno tako, kot če bi na knjigah pisalo "Knjiga" namesto, kaj je v njej. (Ni pa zelo drugače od steklenic v Mercatorju, na katerih preprosto piše "Vino".)

4. **Kratka imena.** Izogibajte se imen, kot so `a`, `aa`, `asd`... ki ne povedo nič. Sicer me boste videli, da bom uporabljal kratka imena na predavanjih, vendar le zato, ker bom tipkal v živo in nimate časa gledati, kako tipkam `imena_nekih_studentov`. Program, v katerem je deset spremenljivk z imeni `a`, `b`, `aa`, `a2`, `a3`, `a4` je nemogoče brati.

5. **Dogovorjena imena spremenljivk.** Po drugi strani pogosto uporabljamo kratka imena za "nepomembne" spremenljivke, ki "živijo" le nekaj vrstic. Kot v matematiki bodo tudi tu spremenljivke z imeni `i`, `j`, `k` pogosto števci, se pravi, nekaj, kar bo teklo od, recimo, 1 do 10. Po eni strani neinformativno, po drugi pa vedno, ko vidimo `i`, `j`, `k` vemo, da gre za števec. `e` bo element kakega zaporedja. `x` in `y` bosta argumenta (tako kot matematiki vedno rečejo `sin(x)`). Imena `s`, `t`, `u` bodo pogosto seznami. Nekateri imajo za seznam raje `xs` (kot angleška množina imena `x`), saj je `xss` potem seznam, ki vsebuje sezname. To boste videli predvsem v nalogah na vajah. Ime `s` bo sicer pogosto predstavljalo tudi besedilo.

Tega se vam niti slučajno ni potrebno zapomniti. Glejte, kako bom pisal na predavanjih in v zapiskih, pa se boste navadili.

Na to temo bi se dalo povedati še veliko in morda nekoč bomo. Za zdaj pa vam resno polagam na srce, kar je napisano tu. Predvsem tistim, ki že znate programirati in se vam zdi najlepše tako, kot delate zdaj. Meni se zdijo v Pythonu lepa Pythonova pravila, v JavaScriptu pa pravila JavaScripta.

Kdor razmišlja drugače, je kot Francoz, ki iz patriotizma namerno govori angleško s francoskim naglasom. Ali Štajerc, ki jo zavija po štajersko.

Vsi našteti dogovori (in še veliko podobnih bomo spoznali sproti) so samo dogovori. Če se jih ne držimo, bodo programi še vedno delovali. Pythonu je vseeno. Ni pa vseeno nam: če se držimo takšnih dogovorov, bodo naši programi preglednejši tako za nas, kot za druge, ki upoštevajo enaka pravila pisanja in bodo morali brati naše programe za nami.

"Vsak, kdor dela, dela tudi napake," se je jež poklapano opravičil krtači. Čeprav nobenega omemba vrednega kosa programa, daljšega od, recimo, 20 vrstic, ne napišemo brez vsaj ene resne napake. Pravzaprav tudi jaz, ki sem kar usposobljen programer, večine časa ne preživim ob programiranju, temveč ob iskanju napak v tem, kar sem pravkar sprogramiral. Začetni tečaji programiranja pa navadno ne posvečajo napakam nobene resne pozornosti. Tu te napake ne bomo ponovili, zato kar takoj naredimo nekaj napak.

Postavimo, najprej, `a` na `7` in izračunajmo `a + b`.

```
a = 7
a + b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-3ef3835d0b71> in <module>
      1 a = 7
----> 2 a + b

NameError: name 'b' is not defined
```

Kadar Python česa ne more storiti, izpiše sporočilo o napaki. Ko bomo programirali zares, bomo videli tudi daljša, ki jih bomo težje razumeli, tole pa je preprosto: `name 'b' is not defined`. Pozabili smo *definirati* `b`, pozabili smo mu dati vrednost.

Kaj pa tole?

```
7 = a
```

```
  File "<ipython-input-13-409be5c547e9>", line 1
    7 = a
      ^
SyntaxError: can't assign to literal
```

Človek, vajen matematike, bi si mislil, da je `a = 7` in `7 = a` eno in isto. V matematiki da, pri programiranju (v normalnih jezikih) pa ne, saj enačaj pomeni prirejanje; v prvem primeru priredimo `a`-ju 7, v drugem primeru pa hočemo sedmici prirediti `a`, kar seveda ne gre. To ima natanko toliko smisla, kot če bi napisali `1 = 2`. (Še več, Python nas bo po prstih celo, če bomo napisali `1 = 1`. Ena je ena, to bo ostala in se ne bo spremenila, niti v ena ne.) Sporočila

o napaki tokrat ne razumemo povsem, saj ne vemo, kaj je "literal", osnovno sporočilo, "can't assign", pa je jasno.

Pridelajmo še eno napako.

```
True = 12
```

```
  File "<ipython-input-14-abb8adb6eb95>", line 1
    True = 12
           ^
SyntaxError: can't assign to keyword
```

Beseda `True` ima poseben pomen in je ni mogoče uporabljati kot spremenljivko. Takšnim besedam pravimo ključne besede, ali, kot bi jim rekel John Kennedy, če bi bil še živ, *keywords*. Tokrat je bil Python še prijazen, pri večini drugih ključnih besed pa ne bo povedal kaj dosti več kot "nekaj je narobe". Poskusimo z dvema, `if` in `in`:

```
if = 7
```

```
  File "<ipython-input-15-9b6cbf7468bb>", line 1
    if = 7
       ^
SyntaxError: invalid syntax
```

```
in = 7
```

```
  File "<ipython-input-16-b1a7fdba72f5>", line 1
    in = 7
     ^
SyntaxError: invalid syntax
```

Sporočilo "invalid syntax" pomeni, da smo napisali nekaj tako čudnega, da Python ne more uganiti, kaj smo mislili in nam lahko le pokaže tisto mesto, na katerem je dokončno obupal nad nami.

Morda je koga zaskrbelo, da nam bodo takšne, rezervirane besede v stalno napoto. Bi se dalo videti spisek? Ne bo hudega. Python 3.7 jih ima 35 (naraščanje je počasno: v zadnjih desetih letih je dobil le dve novi) in zelo hitro bomo mimogrede spoznali in uporabljali skoraj vse. Že od naslednjih predavanj naprej vam ne bo prišlo na misel, da bi uporabili `if` kot ime spremenljivke in še kak teden kasneje vam bo stavek `if = 1` videti grotesken. (Vsaj mene zabolijo oči, ko ga pogledam in prsti se upirajo temu, da bi sploh natipkali kaj takšnega.) Spisek? Ne bom vas strašil z njim, lahko pa si pomagate z Googleom, če hočete. A ni potrebe.

Moj kalkulator ima tudi funkcije

Tako kot spremenljivke, ki v programiranju ne pomenijo čisto istega kot v matematiki in v programiranju ne pomenijo čisto istega, tudi beseda *funkcija* (Rihanna bi rekla *function*, če bi znala programirati) ne pomeni povsem istega. Videti pa je zelo podobno. Imamo, recimo, funkcijo `abs`, ki izračuna absolutno vrednost števila.

```
abs(-2.8)
```

```
2.8
```

Ali pa `pow`, ki naredi isto kot operator `**`.

```
pow(2, 3)
```

```
8
```

Ob `pow` omenimo, da je bolj običajno pišemo `2 ** 3` in ne `pow(2, 3)`. Funkcija `pow` ima skrite moči in uporabljamo jo le takrat, ko jih potrebujemo.

Za razliko od matematikov, ki na funkcijo gledajo kot da *ima določeno vrednost pri določenih parametrih*, računalnik *izračuna vrednost funkcije*, za kar moramo *poklicati funkcijo*. Se pravi, v zadnji vrstici smo *poklicali* funkcijo `pow` in ji *podali* dva *argumenta*, 2 in 3. Funkcija je izračunala vrednost in jo *vrnila*.

Tudi klic funkcije, `pow(2, 3)`, je izraz. Kot katerikoli drugi izraz lahko tudi `pow` in `abs` nastopata kot del izraza.

```
(pow(2, 3) + 2) / 5
```

```
2.0
```

```
pow(2, 3) + abs(-2)
```

```
10
```

In argumenti funkcij so lahko prav tako izrazi.

```
x = 1
yy = pow(abs(-2), x * 3)
yy
```

```
8
```

Funkcije so v resnici zelo pomembna reč. Python ima milijone in milijone funkcij (ne pretiravam, samo en detajl sem zamolčal). Za vsako reč, ki si jo zamislite, obstaja funkcija. Obstaja funkcija, ki bo, če jo pokličete, poslala mail osebi, katere naslov podate kot argument in z vsebino, ki jo podate kot argument. Obstaja funkcija, ki izriše ali pokaže sliko. Funkcija, ki odpre Excel in funkcija, ki piše po njem. S Pythonom lahko naredimo karkoli, le ime funkcije moramo poznati. (Kot rečeno, izpuščam detajle.)

Nizi

Nas še vedno muči radovednost o tem, kakšne podatkovne tipe, poleg številskih, še imamo? Ker radovednost ni lepa čednost, jo bomo najlažje odpravili tako, da jo potešimo. Vsaj malo. ("Skušnjavo najlažje premagaš tako, da ji podležeš," je modroval Oscar Wilde. Res pa je, da so ga potem aretirali in zaprli.) Spoznajmo vsaj še en bolj zapleten podatkovni tip: niz.

Niz ali, po kot mu pravijo Avstralci, *string* (oprostite mi boste morali, da bom pogosto uporabljal angleške izraze; pa mi pokažite, lepo prosim, zidarja, ki ne uporablja vaservage in plajbe temveč vodno tehtnico in svinčnico, pa se bom tudi jaz discipliniral), je zaporedje znakov. Aha, kaj pa je to znak? Znaki so črke, številke, ločila in take stvari. Nize moramo vedno zapreti v narekovaje, bodisi enojne (') bodisi dvojne ("). Uporabiti smemo take, ki so bolj praktični in tudi Python bo izpisoval tako, kot se mu bo zdelo bolj praktično.

```
'Tole je primer niza.'
```

```
'Tole je primer niza.'
```

```
"Tole je pa še en primer niza."
```

```
'Tole je pa še en primer niza.'
```

Tudi ko smo niz zaprli v dvojne narekovaje, je Python izpisal enojne. V resnici mu je vseeno. (V nekaterih jezikih imajo različni narekovaji povsem ali pa vsaj nekoliko različen pomen. V Pythonu ne, pač pa različne pomene, kot bomo videli, dosežemo tako, da prednje postavimo kak primeren znak.)

Tudi nize lahko priredimo spremenljivkam.

```
napoved = "Jutri bosta matematika pa dež"
napoved
```

```
'Jutri bosta matematika pa dež'
```

Celo seštevamo jih lahko.

```
"Jutri bosta " + "matematika" + " pa " + "dež"
```

```
'Jutri bosta matematika pa dež'
```

Ali pa oboje

```
napoved_zac = "Jutri bosta "
mat = "matematika"
dez = "dez"
napoved_zac + mat + " pa " + dez
```

```
'Jutri bosta matematika pa dez'
```

Kako zapleten račun! Predvsem ne spreglejte, da smo dali besedo "pa" pod narekovaje, saj so `napoved_zac`, `mat`, `dez` spremenljivke (ki so v resnici nizi),

`" pa "` pa je niz kar tako. To je nekako tako, kot če bi, ko smo se igrali s številkami, pisali

```
x = 1
y = 3
x + 2 + y
```

```
6
```

V zadnji vrstici sta `x` in `y` sta spremenljivki (ki sta v resnici številki), `2` pa je številka kar tako.

Kaj pa, če bi slučajno pozabili narekovaje?

```
napoved_zac + mat +  pa  + dez
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-a10a1af57195> in <module>
----> 1 napoved_zac + mat +  pa  + dez

NameError: name 'pa' is not defined
```

Jasno? Brez narekovajev je `pa` ime spremenljivke - in to takšne, ki še ni definirana. To je tako, kot če bi namesto

```
ime = "Benjamin"
```

kar je pravilno, rekli

```
ime = Benjamin
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-30-149e124aca6f> in <module>
----> 1 ime = Benjamin

NameError: name 'Benjamin' is not defined
```

Ali pa:

```
napoved = Jutri bosta matametika pa dež
```

```
  File "<ipython-input-31-dd46811d55c1>", line 1
    napoved = Jutri bosta matametika pa dež
                          ^
SyntaxError: invalid syntax
```

Tule računalnik trpi še bolj. Ne le, da so `Jutri`, `bosta`, `matematika`, `pa` in `dež` nedefinirane spremenljivke (ker so pač brez narekovajev),računalnik tudi nima pojma, kaj hočemo pravzapravz početi z njimi, čemu mu naštevamo imena nekih spremenljivk. Jih hočemo sešteti ali zmnožiti ali kaj? Kot vedno, kadar

računalniku napišemo kaj zares zmedenega, nam zastoka le "syntax error" in pokaže mesto, kjer se je dokončno izgubil.

Gornje je tako, kot da bi rekli

```
x = 1
y = 2
z = 3
x y z
```

```
  File "<ipython-input-32-8236878fd6a9>", line 4
    x y z
      ^
SyntaxError: invalid syntax
```

Ubogemu računalniku pač ni jasno, kaj bi z `x`, `y` in `z` ter zakaj mu jih naštevamo.

Tiste, ki so ob vpisu na fakulteto nihali med računalništvom in slavistiko, je nemara zbodlo, da sem pisal "Jutri bo matematika pa dež". Mar ne bi bilo lepše (in bolj prav) "matematika in dež". Ponovite vse, kar smo napisali zgoraj, z `in`, pa boste videli, čemu. Za jezikovno okornost se seveda opravičujem, ampak z `in` tale primer ne bi deloval.

Zakaj pa smo prejle rekli, da uporabimo tiste narekovaje, ki so bolj *praktični*? Čemu bi bili kakšni narekovaji bolj praktični od drugih?

```
"Cesar vpraša nekoliko nevoljen: "Kaj neki?""
```

```
  File "<ipython-input-33-e7b97c04099d>", line 1
    "Cesar vpraša nekoliko nevoljen: "Kaj neki?""
                                      ^
SyntaxError: invalid syntax
```

Ni potrebno biti posebno pameten, da vidimo, kaj ga je (namreč Pythona, ne cesarja) onesrečilo tokrat. Ko vidi prvi narekovaj, ve, da gre za niz. Ko pride do naslednjega narekovaja, se niz, tako méni, niz konča. In potem se seveda zmede, ker nizu sledi nekaj, kar ni podobno ničemur. Zdaj pa poskusimo z enojnimi narekovaji.

```
'Cesar vpraša nekoliko nevoljen: "Kaj neki?"'
```

```
'Cesar vpraša nekoliko nevoljen: "Kaj neki?"'
```

Ker se niz začne z enojnim narekovajem, se bo s takim tudi končal in vsi dvojni narekovaji znotraj niza so samo znaki kot katerikoli drugi - tako kot recimo dvopičje in vprašaj. O tej temi bi lahko napisali še marsikaj, vendar se za zdaj ustavimo.

Iz nizov v števila

Da se reč usede, meditirajmo ob naslednjih vrsticah:

```
a = 1 + 1
b = "1 + 1"
c = "1" + "1"
```

Kakšne so po tem vrednosti spremenljivk `a`, `b` in `c`? Sploh pa, je vse troje legalno ali pa bo Python spet kaj stokal?

V prvo nimamo dvomov, vrednost `a` mora biti enaka 2 (in tudi je). Drugo? `"1 + 1"` je niz; spremenljivki `b` smo priredili niz `"1 +1"`, torej vsebuje ta niz. In ne niza 2? Ne, nihče mu ni naročil, naj poskuša izračunati, koliko je 1 + 1, tako kot pravzprav tudi v `ime = "Benjamin"` ne poskuša izračunati, "koliko" je Benjamin. `"1 + 1"` je niz, kot vsak drugi, čeprav je slučajno podoben računu.

Najbolj zanimivo je tretje. Preden razrešimo vprašanje, se vprašajmo nekaj drugega. Recimo

```
ana = "Ana"
benjamin = "Benjamin"
r = ana + benjamin
```

Kaj dobimo, če seštejemo Ano in Benjamina. (Tončka? Brez duhovičenja, to so resne reči.) Spremenljivka `r` bo imela vrednost `"AnaBenjamin"`. Glede tega smo si menda enotni, ne? (Ako kdo misli, da bomo dobili `"Benjamin Ana"`, saj smo tudi poprej imeli presledke ob oni študijsko-vremenski napovedi, naj pozorno pregleda, kaj smo pisali ondi: vse presledke smo napisali sami.)

No, potem pa vemom: ko seštejemo *niza* "1" in "1" niz "11". "1" in "1" torej ni "2", temveč "11".

Nikar ne zamudimo priložnosti za še eno napako!

```
1 + "1"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-b780703cc5f9> in <module>
----> 1 1 + "1"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Seštevanje je operacija, zato tistemu, kar je levo in desno od `+` pravimo operanda. Sporočilo pravi, da operator `+` ne podpira operandov tipov `int` in `str` (`str` je podatkovni tip, ki predstavlja nize). Dve števili ali dva niza bi znal sešteti, te kombinacije pa ne. Mimogrede, obratni vrstni red da nekoliko drugačno sporočilo:

```
"1" + 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-ec358fc6499a> in <module>
----> 1 "1" + 1

TypeError: can only concatenate str (not "int") to str
```

Stikanje (*concatenation*) je le drugo ime za seštevanje nizov; sporočilo pravi, da ne moremo stakniti niza in števila. Na to napako boste v prvih dveh tednih vaj pogosto naleteli, tako da bodite pripravljeni. Kasneje bomo delali drugačne stvari in bo šlo mimo.

Ker nam bo prišlo vsak čas prav, povejmo, kako iz niza dobimo število. Recimo, torej, da imamo `a = "1"` in `b = "2"`. Radi bi ju sešteli - vendar zares, tako da bomo dobili 3, ne "12". Za to ju moramo najprej (ali pa sproti) spremeniti v števili. Iz niza dobimo število tako, da pokličemo "funkcijo" `int` ali `float`; obe funkciji pričakujeta kot argument niz, ki vsebuje neko število in kot rezultat vrneta celo (`int`) ali realno (`float`) število. (Tole bi se spodobilo in bilo pravično povedati: `int` in `float` v resnici nista funkciji, temveč nekaj drugega, a za potrebe prvih toliko in toliko predavanj, se bomo držali Pythonovega načela "Če hodi kot raca in gaga kot raca, potem je raca", ki bo postalo pomembno proti koncu semestra; če se obnašata kot funkciji, ju bomo brez slabe vesti oklicali za funkciji.)

```
int("42")
```

```
42
```

```
float("42")
```

```
42.0
```

Kar želimo, storimo na tri načine, vsak bo poučen po svoje. Prvi:

```
a = "1"
b = "2"

aa = int(a)
bb = int(b)
aa + bb
```

```
3
```

Naredili smo dve novi spremenljivki, `aa` in `bb`, ki vsebujeta vrednosti `a` in `b`, pretvorjeni v števila. Nato ju seštejemo.

Drugi:

```
a = "1"
b = "2"

a = int(a)
```

```
b = int(b)
a + b
```

```
3
```

Tole je podobno kot prej, le da smo povozili stare vrednosti `a` in `b` z novimi, številskimi, namesto da bi števila zapisovali v druge spremenljivke.

Samo za tiste, ki že znate programirati, a ne v Pythonu: po izkušnjah iz preteklih let, se tule začne polovica predavalnice praskati po glavi, pogumnejši pa se začnejo oglašati, da je Python čuden jezik oziroma, z generaciji lastnejšimi besedami, *kr neki*. To, da tipov spremenljivk ni potrebno deklarirati, so še nekako požrli. Ampak tole tule je pa višek, pravijo: je `a` tipa niz ali število? No, na srečo se lahko skličem na to, kar sem polagal v srce ob prireditvenem stavku: prirejanje priredi imenu neko vrednost. Predvsem pa Python nima spremenljivk v takem pomenu besede, kot ste jih vajeni iz Jave, C# ali C++, temveč ima imena za objekte. Prirejanje `a = "1"` priredi imenu `a` niz "1". Prirejanje `a = int(a)` priredi imenu `a` številko 1. Ime `a` se po tem pač ne nanaša več na nek niz temveč na neko število. Python nima spremenljivk, temveč imena. In imena nimajo tipov.

Tretji:

```
a = "1"
b = "2"

int(a) + int(b)
```

```
3
```

Ker je `int` funkcija, lahko nastopa v izrazu; potrebe, da bi prepisovali številke v kake nove ali stare spremenljivke, niti ni.

Vpis in izpis

Doslej smo z računalnikom komunicirali tako, da smo tipkali ukaze v *ukazno vrstico* in če je imela reč kak rezultat (kot ima izraz `1 + 1` rezultat `2`), ga je računalnik izpisal. Ukazna vrstica nam v Pythonu pogosto pride prav za kakšna preskušanja, ko programiramo zares, pa komunikacija z uporabnikom poteka drugače. Če hočemo, da računalnik kaj izpiše, mu moramo to posebej reči. Se pravi, ko bomo programirali, ne bomo napisali le `1 + 1` temveč "izpiši, koliko je 1 + 1".

Rekli smo, da funkcije pri programiranju niso nekaj takšnega kot funkcije v matematiki: funkcije v matematiki imajo določeno vrednost pri določenih argumentih (še huje, matematiki pravijo, da so funkcije pravilo, ki vsakemu elementu kodomene funkcije določi ... uh, pustimo). "Naše" funkcije pa nekaj delajo in včasih vrnejo kakšen rezultat, recimo številko ali niz ali kaj tretjega. Ena od teh funkcij je namenjena izpisovanju: če jo pokličemo, izpiše tisto, kar smo ji dali kot argument. Imenuje se `print`. Za razliko od, recimo, `abs`, ki zahteva en

argument, namreč poljubno število, in vrne njegovo absolutno vrednost, ali `pow`, ki hoče natanko dva argumenta, lahko damo `print`u poljubno število argumentov - številke, nize ali še kaj tretjega -, pa jih bo lepo izpisala.

```
print(1 + 1, 27, "benjamin")
```

```
2 27 benjamin
```

```
print(napoved_zac, mat, "pa", dez, "in", 18, "stopinj")
```

```
Jutri bosta  matematika pa dez in 18 stopinj
```

Med reči, ki jih izpiše, bo `print`, če ne zahtevamo drugače, postavil presledke.

Druga funkcija, ki nam bo prišla prav, prosi uporabnika, da vpiše kako reč. Kot argument pričakuje niz, vprašanje, ki ga želimo zastaviti uporabniku. Kot rezultat "izračuna" vrne niz, ki ga je vpisal uporabnik.

```
geslo = input("Geslo? ")
```

```
Geslo? skrivnogeslo
```

```
geslo
```

```
'skrivnogeslo'
```

Prvi čisto pravi program

Sestavimo tole: računalnik naj uporabnika prosi za temperaturo v Celzijevih stopinjah in računalnik mu bo izpisal, koliko je to v Kelvinih in koliko v Fahrenheitih. Iz Celzijev dobimo Kelvine tako, da jim prištejemo 273.15, Fahrenheite pa tako, da jih pomnožimo z 9/5 in prištejemo 32 (kogar zanima še kaj, naj pogleda na Wikipedijo.

```
temp_C = input("Temperatura [C]? ")
temp_K = temp_C + 273.15
```

```
Temperatura [C]? 15
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-1f38fa372baf> in <module>
      1 temp_C = input("Temperatura [C]? ")
----> 2 temp_K = temp_C + 273.15

TypeError: can only concatenate str (not "float") to str
```

Eh. Funkcija `input` vrne *niz*, ki ga je vpisal uporabnik. Četudi utegne ta izgledati kot številka, je še vedno niz in k nizom ni mogoče prištevati števil. Kot smo videli, lahko storimo troje: naredimo novo spremenljivko, na primer, `temp_Cf = float(temp_C)`, povozimo staro s `temp_Cf = float(temp_C)` ali pa pretvorbo opravimo kar sproti, tako da računamo `temp_K = float(temp_C)`

`+ 273.15`. Izmed naštetih možnosti se odločimo za četrto in niz pretvorimo, čim ga uporabnik vpiše. Ponovimo torej vso vajo.

```
temp_C = float(input("Temperatura [C]? "))
temp_K = temp_C + 273.15
temp_F = temp_C * 5/9 + 32
print(temp_C, "C je enako", temp_K, "K ali", temp_F, "F")
```

```
Temperatura [C]? 15
15.0 C je enako 288.15 K ali 40.333333333333336 F
```

Vse, kar smo počeli doslej, vključno s tem zgoraj, je kramljanje z računalnikom, To ni pravi program. "Pravi program" je nekaj, kar poženeš in te vpraša, kar te ima vprašati ter izpiše, kar ima izpisati.

Doslej torej nismo bili pravi programerji, temveč le čveke. Zdaj pa končno postanimo pravi programerji. No, recimo. Odprli bomo namreč - Notepad in vanj vtipkali vse, kar smo zgoraj tipkali Pythonu.

Reč shranimo pod imenom, recimo, "temperature.py". (Če to resno kdo dela v Notepadu, naj pri shranjevanju dejansko natipka tudi narekovaje, sicer mu bo Notepad shranil temperature.py.txt.)

Potem odpremo terminal (v Windowsih bo to Cmd, drugje Terminal, shell, Konsole, bash ali kaj takega) in vtipkamo `cd` in ime direktorija, kamor smo shranili temperature.py, potem pa (na Windowsih) `\python35\python temperature.py`, drugje pa `python3 temperature.py`.

S tem, drugim, smo rekli Pythonu, naj *izvede* program, zapisan v datoteki `temperature.py`. Obnaša naj se, kot da bi mu tole tipkali, lepo, vrstico za vrstico.

In to je počel. Najprej je izvedel `input` in nas torej vprašal po temperaturi, v naslednjih dveh vrsticah je računal in v zadnji izpisoval.

Razvojna okolja

Namen gornjega je bil predvsem povedati, da program v Pythonu ni nič drugega kot datoteka z običajnim besedilom (a ne v Wordu - to ni običajno besedilo, saj je datoteka oblikovana tako, da vsebuje še kup drugih podatkov) in da program `python`, ki izvaja programe v jeziku Python, ne dela drugega, kot da prebere takšno datoteko in izvede navodila, napisana v njej.

Pri tem predmetu bomo uporabljali predvsem dva, Thonny in PyCharm, kdor želi, pa lahko uporablja tudi preprostejše urejevalnike.

Preprosti urejevalniki Kdor želi kaj preprostega, bo vzel Sublime ali Visual Studio Code. Drugi je kopija prvega (ukradel mu je približno vse), vendar je odprtokodni, za razliko od Sublimea, ki je sicer zastonj, vendar stalno sitnari, ali ga ne bi morda vseeno kupili.

Oba imata dodatke za veliko jezikov: pomagata nam oblikovati kodo, iskati pozabljena imena spremenljivk v dolgih programih, pokazati dokumentacijo posameznih funkcij in, do neke mere, poganjati programe. Njuna lepota je v njuni univerzalnosti in preprostosti.

Starejši mački prisegajo na starejše urejevalnike, kot so emacs in vi oz. vim. Tudi med študenti se vsako leto najde kakšen in nihče mu ne brani. Tudi mazohizem je v sodobni družbi popolnoma sprejeta orientacija.

Učna okolja Za začetnike so morda primernejša prijaznejša okolja, takšna, ki imajo kaka orodja, ki jasneje kažejo, kaj se dogaja med izvajanjem programov. In to na način, primeren za začetnike - lepo, počasi, nazorno, ne kot profesionalna orodja.

V ta namen bomo uporabljali Thonny. Je res osnoven in preprost ter nima niti vsega, kar ima Visual Studio Code z ustreznim dodatkom. Vendar ima prav tisto, kar bomo potrebovali za ta predmet, vsaj na začetku.

Profesionalna okolja Najbrž najbolj popularno profesionalno okolje za razvoj programov v Pythonu je PyCharm. Gre za komercialno orodje, vendar ga razvija podjetje, ki je zelo naklonjeno tudi odprti kodi, predvsem pa imajo zastonjsko različico, ki ji prav nič ne manjka. Zato izjemoma nimamo slabe vesti, ker študente navajamo na komercialne programe. Predvsem pa je PyCharm je v resnici neverjetno dober, prav tako pa so izjemna tudi druga orodja tega podjetja. (Vsem, ki bi se radi naučili vročega novega jezika priporočam Kotlin.)

PyCharm nam bo pri programiranju pomagal še veliko bolj kot Visual Studio Code. Znal bo dopolnjevati napol napisana imena (nekaj malega zna tudi VS Code, a PyCharm je veliko zvitejši), pomagal pri tipih spremenljivk, argumentih funkcij, iskanju napak...

Odpremo torej PyCharm in naredimo projekt "Programiranje 1". Na levi strani vidimo nekakšne direktorije. Naredimo poddirektorij z zaporedno številko in naslovom predavanj in v njem novo datoteko z imenom temperature.py. (Kako to naredimo, boste že odkrili sami, saj niste prvič za računalnikom.) Poklikamo novo datoteko in vanjo vnesemo program.

Ko je program vtipkan, pritisnemo Ctrl-Shift-F10 v Windowsih ali Ctrl-Shift-R na Os X ali bogvekaj v Linuxu (ali pa v menuju Run izberemo Run, vendar se raje navadite na bližnjice, saj boste tole letos naredili deset tisočkrat) in PyCharm bo pognal naš program. Vpis in rezultati bodo v spodnjem delu okna.

Tako smo spoznali izraze in spremenljivke, se naučili klicati funkcije, vse skupaj zložiti v program in ga izvesti.